Kapitel 2: Grundlagen in Python und Jupyter Notebooks

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 2. Auflage. Sebastopol, CA [u. a.]: O'Reilly.

Überarbeitet: armin.baenziger@zhaw.ch, 2. Januar 2020

Hinweise:

- Dieses Kapitel gibt einen ersten Einblick in Python und das Jupyter Notebook.
- **Es ist somit nicht erforderlich, dass Sie bereits alle Details des Kapitels verstehen.**
- Bedenken Sie, dass man eine Programmiersprache nur lernt, wenn man sich dabei "die Hände schmutzig macht". Versuchen Sie, viel zu üben.

In [1]:
```
%autosave 0
```

```
Autosave disabled
```

`%autosave` ist ein sogenannter *Magic command*, welcher das Jupyter Notebook steuert. `%autosave 100` würde bedeuten, dass das Notebook alle 100 Sekunden gespeichert wird. `%autosave 0` bewirkt, dass das Notebook *nicht* automatisch gespeichert wird.

Als nächstes importieren wir die Bibliothek `sys` und ermitteln Ihre Python-Version:

In [2]:
```
import sys
sys.version
```

Out[2]:
```
'3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.1912 64 bit (AMD64)]'
```

Zellen in Jupyter Notebooks

Die Eingabe von Python-Befehlen bzw. -Funktionen erfolgt in Jupyter Notebooks in sog. Zellen. Output wird gegebenenfalls in der gleichen Zelle dargestellt.

In [3]:
```
4 + 5
```

Out[3]:
```
9
```

Markdown

Text kann mit Markdown-Zellen zu Jupyter Notebooks hinzugefügt werden. Markdown ist eine beliebte Markup-Sprache, die eine Obermenge von HTML ist.

Grundlagen

Text kann *kursiv* und **fett** geschrieben werden.

Listen:

- Man kann einfach Listen bilden, welche auch
  - Unterlisten enthalten können.
  - Das ist sehr praktisch

Nummerierte Listen:

1. Man kann auch nummerierte Listen erstellen.
2. Auch das ist einfach.

Horizontale Linie:

- LaTeX-Formeln im Text, z.B: x^2, geht auch.
- Man kann Formeln auch absetzen:

$$\sum x_i = n \cdot \bar{x}$$

Tabellen: Zudem ist es möglich, Tabellen zu erstellen:

i	x_i	y_i
1	15	32
2	11	21
3	13	25

Link: http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html (http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html)

Kommentare

Kommentare können mit der Raute (schweiz. Gartenhag, engl. hash) hinzugefügt werden. *Alles hinter einer Raute wird vom Interpreter ignoriert.*

In [4]:
```
# Erzeugung einer sog. Python-Liste mit []:
wochentage = ['Mo', 'Di', 'Mi', 'Do', 'Fr', 'Sa', 'So']
# Werte aus der Liste ziehen: Python beginnt die Indizierung mit 0!
wochentage[0]         # erstes Element der Liste
```

Out[4]:
```
'Mo'
```

In [5]:
```
wochentage[1]         # zweites Element der Liste
```

Out[5]:
```
'Di'
```

In [6]:
```
wochentage[-1]        # letztes Element der Liste
```

Out[6]:
```
'So'
```

Kontrollfragen:

In [7]:
```
# Frage 1: Was ist der Output?
wochentage[3]
```

Out[7]:
```
'Do'
```

In [8]:
```
# Frage 2: Was ist der Output?
wochentage[-3]
```

Out[8]:
```
'Fr'
```

Tab-Vervollständigung (Tab Completion)

In [9]:
```
eine_Orange = 'Jaffa'
eine_Banane = 'Cavendish'
```

In [ ]:
```
eine_       # Mit Tabulator vervollständigen
```

In [ ]:
```
# Mit dem Tabulator können auch sog. Methoden und Attribute eines Objekts vervol
lständigt werden.
wochentage.     # Mit Tabulator vervollständigen
```

Introspektion (Object Introspection)

Mit einem Fragezeichen (?) vor oder nach einem Objekt erhält man einige generelle Informationen zum Objekt.

In [ ]:
```
# Informationen zur Liste:
?eine_Orange
```

In [ ]:
```
# Informationen zur Funktion:
?len
```

In [10]:
```
# Anzahl Elemente in der Liste:
len(wochentage)
```

Out[10]:
```
7
```

Magic Commands

- "Magic Commands" starten mit `%` und sind spezielle Befehle, welche nicht in Python selber implementiert sind.
- Mit "Magic Commands" kann man das Jupyter Notebook steuern.
- Wir haben weiter oben bereits ein Beispiel von einem "Magic Commands" gesehen: `%autosave 0`
- Mit `%quickref` erhält man eine Übersicht über die Befehle.
- Ein weiteres Beispiel eines "Magic Command": Definierte Variablen

In [11]:
```
%who
```

```
eine_Banane      eine_Orange      sys      wochentage
```

Grundlagen der Python Sprache

Sprach-Semantik

Einrückung statt Klammern

- Statt (geschweifte) Klammern (wie z.B. in C oder R), werden in Python Programmblöcke durch Einrücken strukturiert.
- Ein Code-Block startet mit einem Doppelpunkt. Danach muss der ganze Block mit *derselben* Anzahl Leerschlägen eingerückt sein.
- *Es ist Usanz, 4 Leerschläge für eine Einrückung zu verwenden*.
- Beispiel (Erklärungen folgen später):

In [12]:
```
x = 2
if x < 0:
    print('Die Zahl', x, 'ist negativ.')
else:
    print('Die Zahl', x, 'ist nicht negativ.')
```

```
Die Zahl 2 ist nicht negativ.
```

- Python Statements müssen *nicht* durch ein Semikolon abgeschlossen werden. Mit Semikolons können aber mehrere Statements auf einer Zeile abgetrennt werden.
- Man sollte von dieser Möglichkeit allerdings nur in Ausnahmefällen Gebrauch machen, da die Lesbarkeit des Codes dadurch erschwert wird.

In [13]:
```
a = 5; b = 6; c = 7    # mehrere Anweisungen auf einer Zeile
print('Summe:', a + b + c)
```

```
Summe: 18
```

Funktionen und Methoden

- Funktionen werden mit runden Klammern und evt. Argumenten aufgerufen.

In [14]:
```
text = 'Datenanalyse'
print(text)
len(text)
# Die Funktion len(x) gibt die Anzahl Elemente des Objekts x zurück.
```

```
Datenanalyse
```

Out[14]:
```
12
```

- Funktionen können verschachtelt werden:

In [15]:
```
print('Das Wort', text, 'hat', len(text), 'Buchstaben.')
```

```
Das Wort Datenanalyse hat 12 Buchstaben.
```

- Optional kann der Funktionswert einer Variable zugewiesen werden.

In [16]:
```
n = len(text)
print(n)
```

```
12
```

- Die meisten *Objekte* in Python haben zugehörige Funktionen, welche Zugriff auf den Inhalt der Objekte haben.
- Man nennt diese Funktionen **Methoden**.
- Methoden werden wie folgt aufgerufen: **`objekt.Methodenname(Argument 1, Argument 2, ...)`**
- Beispiel:

In [17]:
```
text.upper()  # Die Methode upper() wandelt den Text in Grossbuchstaben um.
```

Out[17]:
```
'DATENANALYSE'
```

In [18]:
```
text.count('a')  # Anzahl "a" in der Zeichenkette.
```

Out[18]:
```
3
```

Kontrollfrage:

In [19]:
```
# Gegeben:
z = 'Datenanalyse mit Python'
```

In [20]:
```
# Frage 1: Wie erhalten wir die Länge der Zeichenkette z?
len(z)
```

Out[20]:
```
23
```

In [21]:
```
# Frage 2: Wie viele "e" hat die Zeichenkette z?
z.count('e')
```

Out[21]:
```
2
```

Bibliotheken importieren

Mit dem Befehl `import` können Bibliotheken ("Libraries") geladen werden, welche den Funktionsumfang von Python in bestimmten Dimensionen erweiteren. Beispiel:

In [22]:
```
import numpy as np
```

- Die Anordnung lädt die Bibliothek NumPy, welche eine grundlegende Bibliothek für wissenschaftliche Berechnungen mit Python ist.
- Es ist üblich, die Bibliothek `numpy` mit der Abkürzung `np` zu laden.
- Will man Funktionen dieser Bibliothek aufrufen, stellt man `np.` vor die entsprechende Funktion.
- Beispiel:

In [23]:
```
np.random.seed(321)
np.random.randint(1, 7, 10)
```

Out[23]:
```
array([5, 3, 5, 2, 1, 2, 1, 3, 1, 5])
```

- Die erste Zeile setzt den Seed für die Ausgabe von Quasi-Zufallszahlen. Wenn ein Seed gesetzt wird, erhält man reproduzierbare (immer die gleichen) "Zufallszahlen".
- Die zweite Zeile erzeugt 10 ganze Zahlen (Integers) von 1 bis und mit 6.
- Kapitel 4 wird sich näher mit der Bibliothek `numpy` befassen.
- Später werden wir uns insb. mit der Bibliothek `pandas` beschäftigen.

Binäre Operationen und Vergleiche

```
In [24]: 5 + 7
Out[24]: 12

In [25]: 12 - 21.5
Out[25]: -9.5

In [26]: 4 * 3
Out[26]: 12

In [27]: 9 / 4
Out[27]: 2.25

In [28]: 9 // 4    # Division ohne Rest
Out[28]: 2

In [29]: 9 % 4     # Rest der Division (Modulo)
Out[29]: 1

In [30]: 2 ** 3    # Potenz (Achtung, nicht mit ^ wie z.B. in Excel)
Out[30]: 8

In [31]: 16 ** 0.5 # Potenz mit geborchenem Exponenten (Wurzel)
Out[31]: 4.0

In [32]: a = 2; b = 3

In [33]: a == b    # True, falls a gleich b ist; sonst False
Out[33]: False

In [34]: a <= b    # kleiner oder gleich
Out[34]: True

In [35]: a > b     # grösser
Out[35]: False

In [36]: a != b    # ungleich
Out[36]: True
```

In [37]:
```
(a < 3) & (b < 3)  # "logisches und" mit &
# Nur wenn alle Bedingungen wahr sind, ist der Gesamtausdruck wahr.
```

Out[37]: False

In [38]:
```
(a < 3) | (b < 3)  # "logisches oder" mit |
# Wenn mindestens eine Bedingung wahr ist, ist der Gesamtausdruck wahr.
```

Out[38]: True

Kontrollfragen:

In [39]:
```
# Frage 1: Was ist der Output?
2**3
```

Out[39]: 8

In [40]:
```
# Frage 2: Was ist der Output?
(5 < 4) | (3 != 2)
```

Out[40]: True

Um zu prüfen, ob zwei Referenzen auf das gleiche (unterschiedliche) Objekt(e) zeigen, verwendet man `is` (`is not`).

In [41]:
```
a = [1, 2, 3]
b = a
a is b
# a und b sind identische Objekte (mit zwei Namen).
```

Out[41]: True

In [42]:
```
c = [1, 2, 3]
a is c
# a und b haben gleiche Inhalte, sind aber unterschiedliche Objekte
# (im Speicher).
```

Out[42]: False

In [43]:
```
# Hingegen:
a == c
```

Out[43]: True

In [44]:
```
a is not c
```

Out[44]: True

Modifizierbare und nicht modifizierbare Objekte (Mutable and Immutable Objects)

Die meisten Objekte in Python sind modifizierbar (mutable), so z. B. Listen.

In [45]:
```
eine_Liste = [1, 2, 3]    # Listen sind modifizierbar.
eine_Liste[0]             # Das erste Element der Liste ausgeben.
```

Out[45]: 1

In [46]:
```
eine_Liste[1] = 99        # Das zweite Element der Liste wird verändert.
eine_Liste
```

Out[46]: [1, 99, 3]

Andere Objekte, wie z. B. Tupel, sind *nicht* modifizierbar (*immutable*).

In [47]:
```
ein_Tupel = (1, 2, 3)    # Tupel werden mit runden Klammern erstellt.
type(ein_Tupel)
```

Out[47]:
```
tuple
```

In [48]:
```
ein_Tupel[1]           # Tupel können wie Listen angesprochen werden.
```

Out[48]:
```
2
```

In [49]:
```
ein_Tupel[1] = 99      # Tupel sind nicht modifizierbar.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-4d1ffc34e7c4> in <module>()
----> 1 ein_Tupel[1] = 99      # Tupel sind nicht modifizierbar.

TypeError: 'tuple' object does not support item assignment
```

Kontrollfragen:

In [50]:
```
# Gegeben:
li = [1, 2, 3]
li[1] = 0
```

In [51]:
```
# Frage 1: Was ist der Output?
print(li)
```

```
[1, 0, 3]
```

In [52]:
```
# Frage 2: Was ist der Output?
t = (1, 2, 3)
type(t)
```

Out[52]:
```
tuple
```

Skalar-Typen (Scalar Types)

Die "Ein-Wert-Datentypen" umfassen numerische Typen, Zeichenketten (Strings), Boolean (True, False) und Datum/Zeit.

Numerische Typen

Die primären numerischen Skalar-Typen in Python sind `int` und `float`.

In [53]:
```
ganzzahl = 125 ** 12         # Ganze Zahlen sind vom Typ int.
```

In [54]:
```
type(ganzzahl)
```

Out[54]:
```
int
```

In [55]:
```
dezimalzahl = 7.243        # Dezimalzahlen sind vom Typ float.
type(dezimalzahl)
```

Out[55]:
```
float
```

In [56]:
```
# Wissenschaftliche Notation ist auch möglich:
grosse_zahl = 1.2e6
grosse_zahl
```

Out[56]:
```
1200000.0
```

In [57]:
```
kleine_zahl = 6.78e-3
kleine_zahl
```

Out[57]:
```
0.00678
```

Zeichenketten (Strings)

In [58]:
```
a = 'Eine Möglichkeit, einen String zu schreiben.'
b = "Eine weitere Möglichkeit, einen String zu schreiben"
```

In [59]:
```
a[0]       # Das erste Zeichen des Strings a ausgeben.
```

Out[59]:
```
'E'
```

In [60]:
```
a[-1]      # Das letzte Zeichen des Strings ausgeben.
```

Out[60]:
```
'.'
```

In [61]:
```
a[:8]     # Die ersten 8 Zeichen ausgeben.
```

Out[61]:
```
'Eine Mög'
```

Man nennt die Syntax `a[:8]` *"slicing"*. Wir werden uns damit später ausführlicher befassen.

Zeichenketten sind nicht *modifzierbar (immutable)*.

In [62]:
```
a[0] = 'e'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-9edb2994f20d> in <module>()
----> 1 a[0] = 'e'

TypeError: 'str' object does not support item assignment
```

Man kann aber mit Methoden Zeichenketten ändern bzw. umkopieren:

In [63]:
```
a = 'Dies ist eine Zeichenkette.'
b = a.replace('eine', 'eine kurze')
b
```

Out[63]:
```
'Dies ist eine kurze Zeichenkette.'
```

In [64]:
```
a     # a ist aber nicht modifiziert!
```

Out[64]:
```
'Dies ist eine Zeichenkette.'
```

Mit `+` werden zwei Strings verknüpft.

In [65]:
```
a = 'Dies ist die erste Hälfte'
b = 'und dies ist die zweite Hälfte.'
a + ' ' + b
```

Out[65]: 'Dies ist die erste Hälfte und dies ist die zweite Hälfte.'

Kontrollfragen:

In [66]:
```
# Gegeben:
a = '232'
b = '100'
```

In [67]:
```
# Frage 1: Was ist der Output?
a + b
```

Out[67]: '232100'

In [68]:
```
# Frage 2: Was ist der Output?
b.count('0')
```

Out[68]: 2

Booleans

In [69]:
```
True and True  # and-Bedingung ist wahr, wenn beide Bedingungen wahr sind.
```

Out[69]: True

In [70]:
```
False and True
```

Out[70]: False

In [71]:
```
False or True  # or-Bedingung ist wahr, wenn mindestens eine Bedingung wahr ist.
```

Out[71]: True

In [72]:
```
False or False
```

Out[72]: False

Booleans sind eine Unterklasse von `int` : `False` ist 0 und `True` ist 1

In [73]:
```
True + 2
```

Out[73]: 3

In [74]:
```
True + True + False
```

Out[74]: 2

In [75]:
```
False == 0
```

Out[75]: True

Kontrollfragen:

In [76]:
```
# Gegeben:
a = 0
b = 2
```

In [77]:
```
# Frage 1: Was ist der Output?
a is not b
```

Out[77]: True

In [78]:
```
# Frage 2: Was ist der Output?
(a > 0) + (b > 0)
```

Out[78]: 1

Type-Casting (Typumwandlung)

Mit der Funktion `str` können viele Python-Objekte in einen String umgewandelt werden.

In [79]:
```
a = 7.1
a
```

Out[79]: 7.1

In [80]:
```
s = '7.1'
s
```

Out[80]: '7.1'

In [81]:
```
a*2
```

Out[81]: 14.2

In [82]:
```
s*2         # nicht 14.2, da s ein String ist!
```

Out[82]: '7.17.1'

In [83]:
```
# Mit der Funktion float() wandeln wir das Argument in eine
# Fliesskommazahl um:
float(s)*2
```

Out[83]: 14.2

In [84]:
```
#  Mit int() wird das Argument in eine ganze Zahl umgewandelt:
int(a)
```

Out[84]: 7

None

- `None` ist der "Nichts-Typ" in Python.

In [85]:
```
Vornamen = ['Anna', 'Berta', None]
len(Vornamen)
```

Out[85]: 3

In [86]:
```
Vornamen[2]
# Es erscheint kein Wert, aber auch kein Fehler.
# None ist hier lediglich ein Platzhalter.
```

In [87]:
```
Vornamen[2] = 'Claudia'  # Platzhalter None überschreiben
Vornamen
```

Out[87]: ['Anna', 'Berta', 'Claudia']

Kontrollfragen:

In [88]:
```
# Gegeben:
a = 123
```

In [89]:
```
# Frage 1: Was ist der Output?
a + a
```

Out[89]:
```
246
```

In [90]:
```
# Frage 2: Was ist der Output?
str(a) + str(a)
```

Out[90]:
```
'123123'
```

Verzweigungen und Schleifen

Bedingte Anweisungen und Verzweigungen: if, elif und else

- Eine **Bedingte Anweisung** ist in der Programmierung ein Programmabschnitt, der nur unter einer bestimmten Bedingung ausgeführt wird.
- Eine **Verzweigung** legt fest, welcher von zwei (oder mehr) Programmabschnitten, abhängig von einer (oder mehreren) Bedingungen, ausgeführt wird.
- *Bedingte Anweisungen* und *Verzweigungen* bilden, zusammen mit den Schleifen, die Kontrollstrukturen der Programmiersprachen. Sie gehören zu den wichtigsten Bestandteilen der Programmierung, da durch sie ein Programm auf unterschiedliche Zustände und Eingaben reagieren kann. Quelle: https://de.wikipedia.org/wiki/Bedingte_Anweisung_und_Verzweigung (https://de.wikipedia.org/wiki/Bedingte_Anweisung_und_Verzweigung)

In [91]:
```
x = -3
# Bedingte Anweisung:
if x < 0:
    print('x ist negativ')
```

```
x ist negativ
```

In [92]:
```
x = 2
# Verzweigung:
if x < 0:
    print('x ist negativ')
elif x == 0:
    print('x ist null')
elif 0 < x < 5:
    print('x ist positiv, aber kleiner 5')
else:
    print('x ist mindestens 5')
```

```
x ist positiv, aber kleiner 5
```

Kontrollfrage:

In [93]:
```
# Gegeben:
a = b = 6
```

In [94]:
```
# Frage: Was ist der Output?

if a <= b:
    print('eins')
else:
    print('zwei')
```

```
eins
```

for-Schleife (loop)

for-Loops iterieren über eine Kollektion (z.B. Liste, Tupel) oder einen Iterator (siehe unten). Die Standard-Syntax lautet:

```
for wert in Kollektion:
    # mach etwas mit wert
```

Beispiele:

In [95]:
```
liste = [0, 1, 2, 3, 4]
for zahl in liste:
    print(zahl ** 2)
```

```
0
1
4
9
16
```

In [96]:
```
# Beispiel mit Interator:
# range(n) erzeugt den Iterator 0, 1, 2, ..., n-1

for zahl in range(5):
    print(zahl ** 2)
```

```
0
1
4
9
16
```

Loops können auch verschachtelt werden:

In [97]:
```
for i in ['A', 'B']:
    for j in ['a', 'b', 'c']:
        print(i + j)
```

```
Aa
Ab
Ac
Ba
Bb
Bc
```

Kontrollfrage:

In [98]:
```
# Gegeben:
zahlen = [1, 2, 3, 4]
```

In [99]:
```
# Frage: Bestimmen Sie die Quadratwurzeln der vier Zahlen
# der Liste "zahlen".

for zahl in zahlen:
    print(zahl ** 0.5)
```

```
1.0
1.4142135623730951
1.7320508075688772
2.0
```

range

Wie zuvor gesehen, kann mit `range` eine Zahlenfolge erstellt werden.

In [100]:
```
range(10)
```

Out[100]:
```
range(0, 10)
```

In [101]:
```
list(range(10))    # Mit der Funktion list() wird eine Liste generiert.
```

Out[101]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [102]:
```
list(range(0, 20, 2))      # gerade Zahlen von 0 bis (aber ohne) 20
```

Out[102]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

In [103]:
```
list(range(5, 0, -1))      # ganze Zahlen von 5 bis (aber ohne) 0 (zurückgezähl
t)
```

Out[103]:
```
[5, 4, 3, 2, 1]
```

`range` wird oft als Iterator verwendet, wie zuvor gesehen.

In [104]:
```
# Noch ein Beispiel:
for i in range(6):
    print('Die Zahl ist', i)
```

```
Die Zahl ist 0
Die Zahl ist 1
Die Zahl ist 2
Die Zahl ist 3
Die Zahl ist 4
Die Zahl ist 5
```

Kontrollfragen:

In [105]:
```
# Frage 1: Erstellen Sie eine Liste mit den ungeraden
# natürlichen Zahlen kleiner 30:
liste = list(range(1, 30, 2))
liste
```

Out[105]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

In [106]:
```
# Frage 2: Erstellen Sie die Liste [30, 27, 24, ..., 3, 0]:

liste = list(range(30, -1, -3))
liste
```

Out[106]:
```
[30, 27, 24, 21, 18, 15, 12, 9, 6, 3, 0]
```

Fazit

- Damit sind wir am Ende der etwas längeren Einführung angelangt.
- Wir haben einige wichtige Grundlagen von Python kennen gelernt.
- Wie bereits in der Einleitung erwähnt, werden viele der hier erstmals vorgestellten Konzepte später im Kurs genauer erläutert.
- **Speichern Sie das Notebook (Icon ganz links in der Toolbar). Danach sollten Sie das Notebook über das Menü `File > Close and Halt` schliessen. (Sie sollten nicht einfach den Browser schliessen.)**